which the SVC was executed. This record describes the remediation behavior to be executed.

The RPDF consists of a header record, followed by one patch data record for each instruction patch in the NCDF. The header record contains:

The library and load module name

The number of sub-records

Space reserved for SVC usage.

Each patch data record contains:

The patch type

Control Section (C-SEC) offset of the patched instruction (the address of the SVC)

Parameter data which the SVC, uses to execute the patch

When SVC starts running, it loads a library of patch modules. There is one of these modules for each type of patch that can be applied. When SVC needs to execute a patch, it calls the patch module specified by the RPDF record, passing a parameter block which contains information determined at analysis time. The library module then executes the patched code, and returns to SVC, and then to the remediated program.

After the necessary modifications are analyzed, the modifications necessary to the load module to provide the modified functionality as described above are thus defined **120**. The defined modified load module **124** is then saved **122**. The modified load module **124** may include an external reference table for trapping date exceptions or otherwise controlling program execution, as well as modified and/or additional object modules. For example, additional load modules may provide input and output data reformatting and/or transformation using a sliding window technique, which might possibly alleviate the need for modifications to the original object modules within the load module **108**.

The stored modified load module **124** is subsequently executed by evaluation of the JCL referencing command **126** in the normal sequence of program execution. It is noted that the modified load module **124** may be generated in advance of use or the entire procedure applied immediately prior to program execution. After execution of the modified load module **126**, normal program execution continues, indicated by the stop **128**, which may include execution of other load modules, modified load modules, or the modification and execution according to the present invention of other load modules.

There has thus been shown and described novel object code remediation systems, which fulfill all the objects and advantages sought therefor. Many changes, modifications, variations, combinations, subcombinations and other uses and applications of the subject invention will, however, become apparent to those skilled in the art after considering this specification and the accompanying drawings which disclose the preferred embodiments thereof. All such changes, modifications, variations and other uses and applications which do not depart from the spirit and scope of the invention are deemed to be covered by the invention, which is to be limited only by the claims which follow.

What is claimed is:

1. A method for automatically modifying computer program logic with respect to a selected data type, comprising the steps of:

(a) analyzing object code representing computer program logic from the computer program to identify references to the selected data type, substantially without reference to or reconstruction of source code, wherein the object code representing computer program logic is analyzed by one or more processes selected from the group consisting of:

disassembly, further comprising the step of applying inferential analysis and state dependent analysis to the disassembled object code representing computer program logic,

scanning data files referenced by the object code representing computer program logic to locate data formatted as date data, and

tracing presumed references to the selected data type through a logical flow of the computer program logic;

(b) modifying the computer program logic with respect to the selected data type to alter computer program logical execution with respect thereto; and

(c) storing information representing the modified computer program logic for execution.

2. The method according to claim **1**, wherein the object code representing computer program logic is modified by substituting elements, substantially without changing a length or arrangement of the object code.

3. The method according to claim **1**, wherein the object code representing computer program logic is modified by replacement of an instruction with a subroutine call instruction.

4. The method according to claim **3**, further comprising the steps of replacing an instruction of the object code with a no-operation code.

5. The method according to claim **1**, wherein the object code is modified by replacing an instruction of the object code with a jump instruction.

6. The method according to claim **1**, wherein the object code is modified by replacing an instruction of the object code with a trap instruction.

7. The method according to claim **1**, wherein the object code is modified by replacing an instruction of the object code with a new machine instruction.

8. The method according to claim **7**, wherein the new instruction is a data-dependent instruction, having at least two functions selectively executed depending on a value of stored data.

9. The method according to claim **1**, wherein modified program logic is represented in a separate object code module from the object code representing computer program logic.

10. The method according to claim **1**, wherein the object code representing computer program logic is modified by a patch.

11. The method according to claim **1**, wherein the object code representing computer program logic is analyzed by a process comprising disassembly, further comprising the step of applying inferential analysis and state dependent analysis to the disassembled object code representing computer program logic.

12. The method according to claim **1**, wherein the selected data type comprises date-related information.

13. The method according to claim **1**, wherein the computer program logic executes on an IBM mainframe compatible computer.

14. The method according to claim **1**, wherein the computer program logic executes under an operating system selected from the group consisting of IBM MVS, VM, OS400 and OS2.

15. The method according to claim **1**, wherein said analysis is automated.

16. The method according to claim **1**, wherein the object code representing computer program logic is included in a load module, each load module also including a load map defining a computing environment of the object code.

17. The method according to claim **16**, wherein a sequence of load module execution is defined by a series of Job Control Language statements.